Manual de Usuario

AplicaTem v3.5

Desarrollado por: **TSU Jesús Bolívar.**

28 de Enero de 2010

Email: Bolivar535@hotmail.com

Telf. 0424-8106551 / 0293-4188693

Tabla de Contenidos

1. Bienvenidos a AplicaTem 3
1.1 Introducción 3
1.2 ¿Qué es un Sistema de Gestión de Contenidos (CMS)? 3
1.3 Objetivos del Manual de Usuario 3
1.4 AplicaTem para desarrolladores 3
2. Instalación y Configuración 5
2.1 Requisitos del Sistema 5
2.2 Instalación 5
2.3 Configuración 5
3. Descripción de los Elementos Básicos de AplicaTem 7
3.1 Organización de archivos 7
3.2 Plantilla/Temas 9
3.3 Componentes 10
3.4 Librerias 11
4. Administrador de Modulos de AplicaTem 13
4.1 administrar Modulos 13
4.2 Agregar objetos de formulario a los modulos 14
4.3 Administrar Perfil 16
4.4 Limpiar Modulos 17
5. Generador de Reportes de AplicaTem 18
5.1 Generar Reportes 18
6. Proceso de carga de formularios 21
6.1 Cargar Formularios 21
6.2 Archivo formulario.php 22
7. Porceso de Restrinción de Usuario 24
7.1 Proceso iniciar Sesion 24
7.2 Archivo Usuario.php 24
7.3 Proceso Cargar Modulos 25
8. Proceso de validación de datos 27
8.1 Validar datos 27
8.2 Archivo validacion.js 27
9. Proceso de Administración de clases 29
9.1 Administrar clases 29
9.2 Archivo administrar.php 30
10. Proceso Validación de Existencia de Datos 33
10.1 Validar existencia de un campo 33
10.2 Archivo validarExistencia.php 33
11. Proceso de carga de Información 36
11.1 Cargar Información 36
11.2 Archivo informacion.php 36
12. Proceso de Visualizar Reportes 38
12.1 Visualizar Reportes en HTML 38
12.2 visualizar Reportes en PDF 40

1. Bienvenidos a AplicaTem

1.1 Introducción

AplicaTem es un pequeño sistema de gestor de contenido que te permite administrar el código de la mayor parte de tu proyecto bajo ambiente Web, llevándolo desde un nivel de diseño, a un nivel de programación, a través de una interfaz en HTML sencilla y manejable. Donde automáticamente te crea los formularios, las clases, la base de datos, los reportes y a la vez puedes incluir, modificar, eliminar y consultar datos sin necesidad de visualizar el código fuente.

1.2 ¿Qué es un Sistema de Gestión de Contenidos (CMS)?

Un CMS es un sistema de software para ordenador que permite organizar y facilitar la creación de documentos y otros contenidos de un modo cooperativo. Con frecuencia, un CMS es una aplicación Web usada para gestionar sitios Web y contenidos Web.

En el caso de AplicaTem te facilita la creación y organización de clases, formulario, reportes etc.

1.3 Objetivos del Manual de Usuario.

Los objetivos del Manual del Usuario son:

- Ayudarle a instalar y configurar AplicaTem!
- Proporcionar una guía sobre como utilizar los módulos propio de la aplicación
- Describir los diferentes componentes y módulos de la aplicación.
- Proporcionar instrucciones detalladas en cuanto a los procesos que se llevan a cabo.

Es un Manual Técnico en el que se profundiza sobre la modificación, diseño y desarrollo de los diferentes elementos de AplicaTem! o cualquier tipo de programación. No se darán detalles sobre PHP, MySQL o cualquier software esencial sobre el que necesita disponer ciertos conocimientos.

1.4 AplicaTem para desarrolladores.

AplicaTem incluye:

- Código HTML
- Código CSS
- Código JAVASCRIPT
- Tecnología AJAX
- Código PHP
- Código SQL, en diferentes administradores de base de datos (MySQL, POSTGRESQL,).

AplicaTem trabaja con:

- Componente de Base de Datos; **DataAccess,** que incluye: DataAccess.php, DataBaseMetodos.php, MySql.php, Postgres.php, SqlServer.php.

- Componente de Seguridad; **SecurityAccess,** que incluye: Securidad.php, Usuario.php, Modulo.php, Perfil.php, ModuloPerfil.php.
- Librería Epoch para mostrar el calendario de fechas
- Librería Eyedatagrid para listar los datos en tablas.
- Librería FPDF para generar reporte en PDF.

AplicaTem permite:

- Seleccionar la base de datos con la cual desea a trabajar (MySql, Postres), crear y restaurar la base de datos por defecto de AplicaTem.
- Administrar módulos, donde se crea los formularios, las clases y las tablas.
- Administrar perfiles de usuarios asignándole módulos y privilegios.
- Gestionar usuarios de la aplicación.
- Administrar objetos de formulario y definir sus propiedades (campo de texto, área de texto, lista-menú, grupo de opciones, casilla de verificación, campo oculto, campo password, y campo de archivo).
- Limpiar código fuente.
- Crear DataPicker para mostrar las fechas en calendario.
- Crear DataGrid para listar los datos en tablas.
- Generar Reportes en HTML y PDF.
- Seleccionar Plantillas.
- Seleccionar Temas.

2. Instalación y Configuración

2.1 Requisitos del Sistema

Antes de descargar el software AplicaTem!, debe asegurarse que su servidor Web, cumple los requisitos mínimos para utilizar AplicaTem!. Son los siguientes:

- PHP 5.2.x o superior - http://www.php.net
- MySQL 5.0 o superior - http://www.mysql.com
- Apache 2.2.4 o superior - http://www.apache.org

Además debe comprobar que el módulo PHP tenga instalado el soporte para MySQL, POSTGRES. AplicaTem! puede utilizarse con los principales navegadores Web, incluyendo: Firefox, Internet Explorer (versión 7.0+) y Google Crome. Estos navegadores se aprovechan de la interfaz Administrativa de AplicaTem.

2.2 Instalación

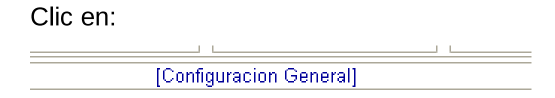
Descomprima los archivos en un directorio situado en la raíz del servidor Web. Si usa un servidor Apache, habitualmente será en c:/apache/groupapache/htdocs/, en un paquete de instalación c:/appserv/www/ o c:/wamp/www/; pero esta ubicación puede variar. Presuponemos que dispone de un servidor Web en marcha y que conoce donde colocar los archivos para que sean visualizados en el navegador.

2.3 Configuración.

Para configurar la aplicación con un manejador de base de datos puede hacerlo a través de la interfaz de usuario o editando el archivo de configuración.

A través de la interfaz de usuario:

Clic en:



La aplicación cargara el formulario correspondiente para la configuración.

Configuración de la Base de Datos

Manejador	○ PostgreSQL ◉ MySQL
Servidor	localhost
Usuario	root
Contraseña	••••
Base de Datos	aplicatem
☑ Restaurar Base de Datos si no existe	
REGISTRAR	

Introduce los datos correspondientes a la configuración, opcionalmente puedes restaurar la base de datos por defecto, clic en el botón REGISTRAR.

Editando el archivo de configuración:

Primero que nada se debe crear y restaurar la base de datos en su manejador de base de datos preferido. Para ello dispone de un respaldo, para Postgres en **/DataBase/DataBasePostgres.backup** y para MySQL en **/DataBase/DataBaseMySql.sql**.

Abrir el achivo /**DataBase/Acceso.php**

```php
<?php
	//PERMITE CREAR LA CONEXION CON LA BASE DE DATOS EN UN MANEJADOR DETERMINADO
	require_once("DataAccess.php");
	//NOMBRE DEL MANEJADOR DE LA BDD
	$manejador='Postgres';
	//NOMBRE DEL HOST DONDE SE ENCUENTRA EL SERVICIO WEB Y DE LA BDD
	$host='localhost';
	//NOMBRE DE USUARIO QUIEN CONECTA CON LA BDD
	$usuario='postgres';
	//CLAVE DE USUARIO QUIEN CONECTA CON LA BDD
	$clave='123456';
	//NOMBRE DE LA BDD A CONECTARSE
	$data base='aplicatem';
```

Cambiar los valores establecidos por defecto por los de tu configuración y guardar los cambios.

La aplicación viene con un usuario por defecto para poder entrar en la aplicación con un perfil de *Administrador*

Usuario: *admin.*
Contraseña: *admin.*

3. Descripción de los Elementos Básicos de AplicaTem

3.1 Organización de archivos.

Los archivos de la aplicación están organizados en carpetas para el mayor entendimiento.

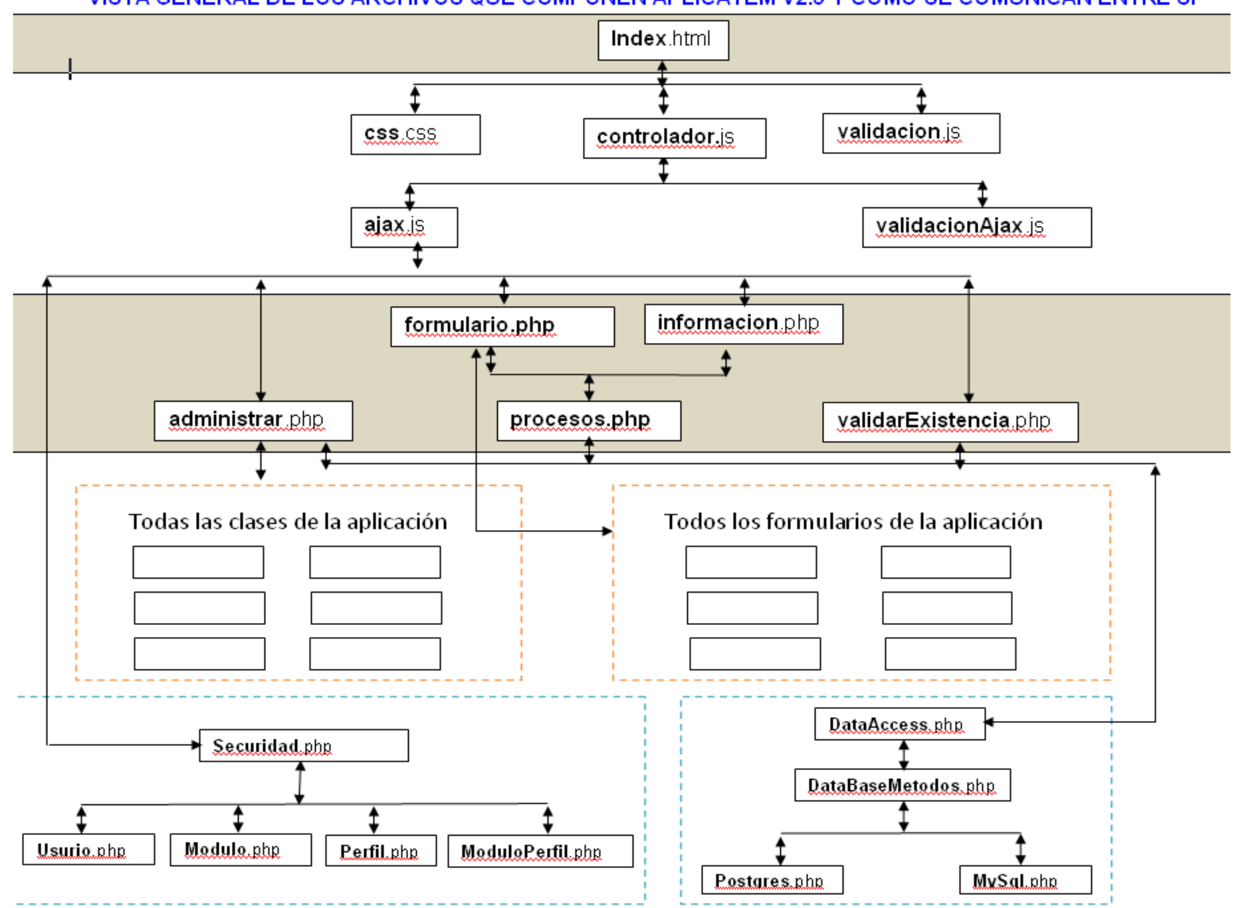
Listado de Carpetas.

- **DataBase/** donde se encuentras todos los archivos del componente de base de datos incluyendo los archivos de respaldo en diferentes manejadores de base de datos. Mas información **ver Sección 3.3 Componente de base de datos**
- **Seguridad/** donde se encuentras todos los archivos del componente de seguridad. **ver Sección 3.3 Componente de Seguridad**
- **javascript/** donde se encuentras todos los archivos javascript.
  - **controlador.js** contiene los procesos en javascript más importantes de la aplicación.
  - **ajax.js** el archivo que controla en envío y la recepción de datos del servidor a través de ajax.
  - **validacionAjax.js** para asignar datos encontrados a los formularios.
  - **validacion.js** contiene todos las validaciones de datos.
- **Clases/** es la carpeta donde se almacenan las clases creadas por la aplicación.
  - **Persona.php** clase de ejemplo para la administración de personas.
- **Formulario/** es la carpeta donde se almacenan todos los formularios correspondientes a las clases.
  - **Sesion.php** contiene el formulario en HTML para iniciar sesión.
  - **Usuario.php** contiene el formulario del modulo usuario
  - **Perfil.php** contiene el formulario del modulo perfil.
  - **Modulo.php** contiene el formulario para administrar módulos
  - **Persona.php** contiene el formulario del modulo Persona.
  - **Configuracion.php** contiene el formulario para configurar el manejador de base de datos
- **estilos/** donde se encuentra el estilo por defecto de la aplicación.
  - **css.css** contiene el estilo por defecto de la aplicación.
- **imagenes/** donde se almacenan las imágenes mostradas en la aplicación.
- **procesos/** por defecto vacía, para almacenar algunos archivos de procesos que se requieran.
- **include/** para almacenar todas las librerías a utilizar en la aplicación.
  - **Enoch**/ librería utilizada para la creación de calendarios para las fechas
  - **eyedatagrid/** librería utilizada para la creación de DataGrid.
  - **FPDF/** librería utilizada para la creación de reportes en PDF.

Reportes/ por defecto vacía, para almacenar todos los reportes, tanto en HTML como en PDF.

Afuera en la carpeta principal se encuentra los archivos que contienen los procesos principales de la aplicación.

- **index.html:** archivo principal que contiene definida la interfaz por defecto de la aplicación
- **formulario.php:** Realiza las llamadas a los formularios. Ver sección:***6.2 Archivo formulario.php***
- **informacion.php:** para gestionar información en php. Ver sección: ***11.2 Archivo informacion.php***
- **procesos.php:** contiene algunos procesos importantes en php.
- **administrar.php:** para crear objetos de clases y llamada a métodos. Ver sección: ***9.2 Archivo administrar.php***
- **validarExistencia.php:** permite realizar consultas a la base de datos y retornar los resultados. Ver sección: **10.2 *Archivo validarExistencia.php***



3.2 Plantillas/Temas.

La interfaz por defecto esta compuesta por el archivo **/index.html** y el estilo **/estilos/css.css**



La interfaz esta basada en una maquetación de tres columnas, detalles a continuación:

Banner: para colocar un banner de la aplicación

Cabecera: para colocar algunos enlaces públicos

Barra de sesión: para mostrar información de la sesión

Lateral Izquierdo: para listar los módulos una vez iniciado sesión

Principal: para mostrar el contenido a procesar

Lateral derecho: para mostrar algunos enlaces externos.

Pie: para mostrar información de la aplicación así como Créditos y Licencia

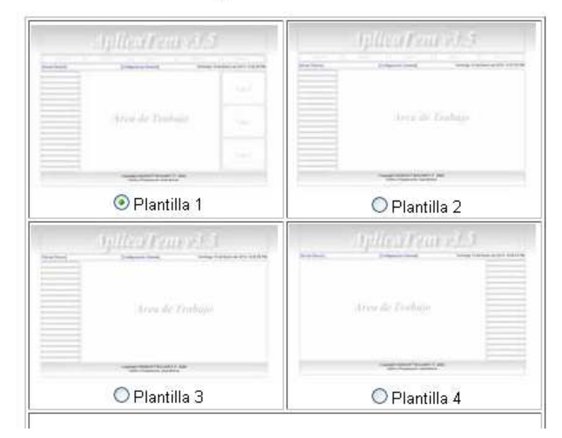
NOTA: Cabe destacar que AplicaTem soporta la selección de plantillas y Temas para ello entre en:

[Configuracion General]

Luego en Plantillas o Temas

Plantillas
Temas

Configuración de Plantilla



Configuración de Tema



Para agregar una plantilla o tema solo tienes que seleccionarlo y darle Clic en **APLICAR**

Ejemplo de los estilos en código HTML

```
<div id="contenedor">
    <div id="baner">
    </div>
    <div id="cabecera">
    </div>
    <div id="reloj">
        <span id="usuario"></span>
        <span id="sesion"></span>
        <span id="tiempo"></span>
    </div>
    <div id="cuerpo">
        <div id="lateral">
        </div>
        <div id="otrolado">
       </div>
        <div id="principal">
        </div>
    </div>
    <div id="pie">
    </div>
</div>
```

3.3 Componentes.

AplicaTem trabaja con dos componentes importantes:

Componente de Base de Datos

Es un componente que te permite migrar un proyecto de una base de datos a otra sin necesidad de modificar en varias partes del código, basta con configurar la conexión con otro Manejador de base de datos para ponerlo a funcionar.

Este componente incluye los archivos:

- **Acceso.php:** es el archivo de configuración.
- **DataAccess.php:** clase que se encarga de llamar dependiendo de un parámetro (Postgres/MySql) a la clase que contiene los métodos de base de datos.
- **DataBaseMetodos.php:** es la interfaz que define los métodos para realizar la conexión y las consultas a la base de datos.
- **Postgres.php:** Clase que contiene los métodos de conexión y consulta para PostgreSQL.
- **MySql.php :** Clase que contiene los métodos de conexión y consulta para MySQL.

Componente de Seguridad

Es un componente que se encarga de la seguridad de la aplicación tanto para el acceso a ella, como para definir diferentes niveles de usuario.

Este componente incluye los archivos:

- **Seguridad.php:** archivo que se encarga de redireccionar una llamada a otro archivo dependiendo de una acción
- **Modulo.php:** clase para administrar los módulos de la aplicación
- **Perfil.php:** clase para administrar los Perfiles de la aplicación
- **ModuloPerfil.php:** clase para administrar los diferentes módulos que va a tener un perfil y contiene los permisos (incluir, modificar eliminar) que tiene un perfil para un módulo. especifico
- **Usuario.php:** clase que se encarga de administrar los usuarios de la aplicación, así como también contiene los métodos para iniciar sesión, cerrar sesión y cargar los módulos dependiendo del perfil que tenga el usuario.

3.4 Librerias.

AplicaTem implementa varias librerías listadas a continuación:

Librería Epoch

Es una librería para mostrar calendarios de fechas.

- *Epoch DHTML JavaScript Calendar - Version 2.0.2*
- *English Edition*
- *Primary JavaScript File*

- *(c) 2006-2007 MeanFreePath*
- *Free for NON-COMMERCIAL use - see website for details and updates*
- *http://www.meanfreepath.com/javascript_calendar/index.html*

Librería Eyedatagrid

Es una librería para mostrar DataGrid, nos ofrece una organización en tablas para listar los datos.

- *EyeDataGrid*
- *Provides datagrid control features*

- *LICENSE: This source file is subject to the BSD license*
- *that is available through the world-wide-web at the following URI:*
- *http://www.eyesis.ca/license.txt. If you did not receive a copy of*
- *the BSD License and are unable to obtain it through the web, please*
- *send a note to mike@eyesis.ca so I can send you a copy immediately.*

- *@author Micheal Frank <mike@eyesis.ca>*
- *@copyright 2008 Eyesis*
- *@license http://www.eyesis.ca/license.txt BSD License*
- *@version v1.1.6 12/3/2008 10:04:44 AM*
- *@link http://www.eyesis.ca/projects/datagrid.html*

Librería FPDF

Es una librería para crear reportes en PDF

- *Software: FPDF*
- *Version: 1.53*
- *Date: 2004-12-31*
- *Author: Olivier PLATHEY*
- *License: Freeware*
- *You may use and modify this software as you wish.*
- *www.fpdf.org/*

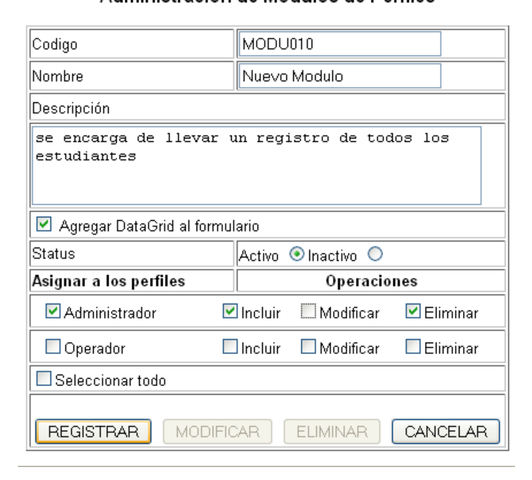
4. Administrando de Modulos de AplicaTem

4.1 Administrar Modulos.

Para registrar nuevos módulos a través de AplicaTem debes iniciar sesión [Iniciar Sesion] con una cuenta de administrador.

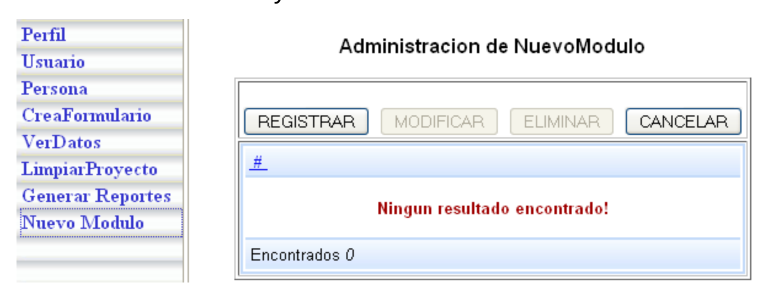
Clic en el modulo, Modulo

La aplicación cargara el formulario correspondiente.



Se debe colocar el nombre del modulo, una descripción para ese modulo y opcionalmente si desea agregar un datagrid al formulario para ir listando los datos al mismo tiempo que se administran. En la parte de abajo se carga una lista con todos los perfiles creados donde se debe seleccionar que perfil va a tener acceso a este modulo y que operaciones podrá realizar.

Una ves registrado se debe recargar la página para que se pueda cargar en la lista de módulos el nuevo modulo creado y al darle clic se mostrará el formulario vacío.



A nivel de usuario ya el nuevo modulo esta creado. Se pueden verificar los cambios realizados en el código fuente:

- Se creó un nuevo archivo ***/Clases/NuevoModulo.php*** que representa la clase NuevoModulo.
- Se creó un nuevo archivo ***/Formulario/NuevoModulo.php*** que representa El formulario NuevoModulo.

- En el archivo **/formulario.php** se agrego una nuevo case NuevoModulo para llamar al formulario
- En el archivo **/javascript/controlador.js** se agrego una nuevo case en el método verificar() que es llamado cuando se desea administrar la tabla nuevomodulo
- En el archivo **/javascript/validacionAjax.js** se agrego un nuevo case en el método **asignarCampos()** que es llamado cuando se consulta un registro en la tabla nuevomodulo y asigna los valores a los campos de formulario.
- Se creo la tabla nuevomodulo en el manejador de base de datos seleccionado.
- Si activo la opción para agregar datagrid al formulario se creo un archivo **/procesos/datagrid_NuevoModulo.php** para listar los datos en el datagrid

4.2 Agregar objetos de formulario a los modulos.

Para agregar objetos de formulario a los módulos a través de AplicaTem, debes iniciar sesión con una cuenta de administrador.

Clic en el modulo, CreaFormulario

La aplicación cargará el formulario correspondiente

Administración de formularios del Sistema



Se debe seleccionar de la lista el modulo al cual se desea administrar objetos de formulario y dar clic en el botón ***Editar Objetos.***

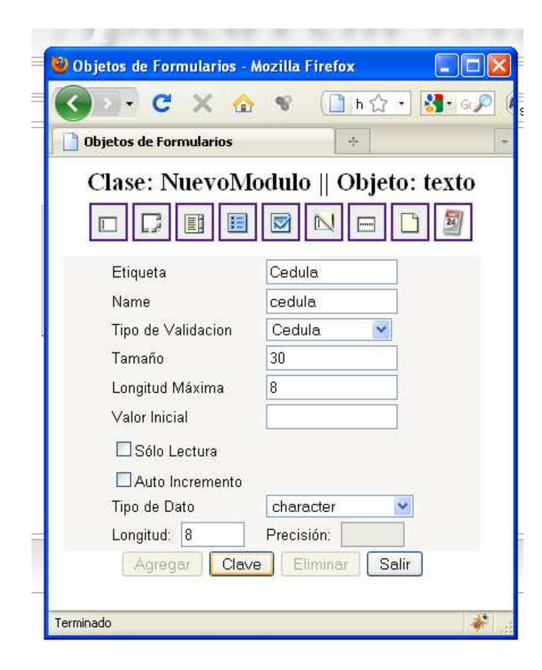
La aplicación cargara una ventana secundaria para administrar los objetos de formulario.



Se debe seleccionar algún objeto de formulario de la barra superior.

Campo de texto, área de texto, lista, grupo de opción, casilla de verificación, campo oculto, campo password, campo de archivo, fecha datapicker.

Una vez seleccionado la aplicación cargará el formulario para ese objeto.



En el formulario se deben llenar los campos Que representan las propiedades de los formularios y las columnas de la tabla *nuevomodulo*.

Hay una lista desplegable de todas las validaciones de datos por defecto que tiene definida la aplicación.

Así como también una lista de los principales tipos de datos utilizados en una tabla.

Opcionalmente puedes agregar al campo clave auto incremento.

NOTA: el primer objeto registrado se agregara como campo clave.

Una ves registrado todos los objetos de formulario se debe salir y automáticamente la aplicación recargará la pagina para que puedan hacer efecto los cambios, al darle clic en el modulo ***Nuevo Modulo*** se mostrará el formulario con los objetos de formularios creados y el datagrid en caso de haberlo seleccionado.



A nivel de usuario los objetos de formularios ya están creados, una vez aquí, la aplicación esta capacitada para administrar (incluir, modificar o eliminar) cualquier registro de este modulo.

Se pueden verificar los cambios realizados en el código fuente para cada uno de los objetos:

- En el archivo ***/Clases/NuevoModulo.php*** que representa la clase NuevoModulo, se agregaron los atributos privados y los métodos públicos, así como también los métodos para incluir, modificar y eliminar registros de esta tabla.
- En el archivo ***/Formulario/NuevoModulo.php*** que representa el formulario NuevoModulo, se agregaron los objetos de formularios en formato HTML e indentados para mayor entendimiento.
- En el archivo **/javascript/controlador.js** en el método **verificar(), case "NuevoModulo"**: se agregaron los nuevos objetos tanto para validarlos, como concatenados listos para enviarlos.
- En el archivo **/javascript/validacionAjax.js** en el método **asignarCampos()** en el **case "NuevoModulo"** se agregaron los objetos de formularios para asignarles los campos cuando se realice la consulta.
- Se agregaron las nuevas columnas a la tabla **nuevomodulo** en el manejador de base de datos seleccionado.

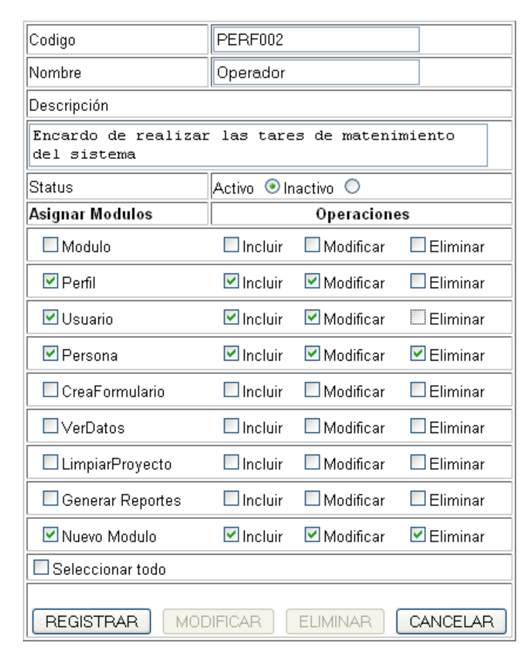
4.3 Administrar Perfil.

Para agregar, modificar o eliminar perfiles de usuarios de la aplicación, debes iniciar sesión con una cuenta de administrador.

Clic en el modulo, Perfil

La aplicación cargará el formulario correspondiente

Administracion de Perfiles de Usuarios



El código se carga automáticamente.

Se deben Completar los datos correspondientes como: el nombre, descripción y el status.

La aplicación cargara automáticamente el listado de todos lo módulos registrados donde se podrán seleccionar los módulos que va a tener acceso este nuevo perfil y las operaciones que podrá realizar.

NOTA: En este proceso no se alteran los archivos de la aplicación.

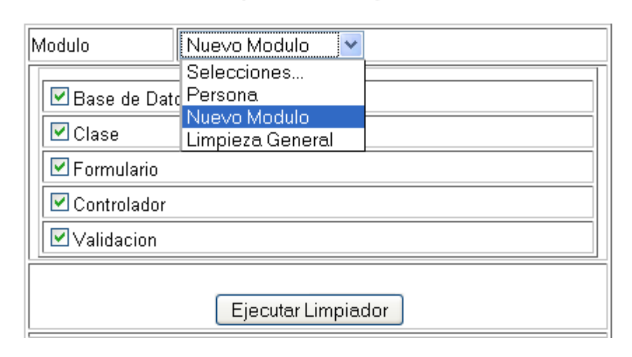
4.4 Limpiar Modulos.

Para agregar, modificar o eliminar perfiles de usuarios de la aplicación, debes iniciar sesión con una cuenta de administrador.

Clic en el modulo, LimpiarProyecto

La aplicación cargará el formulario correspondiente

Limpieza de Proyecto

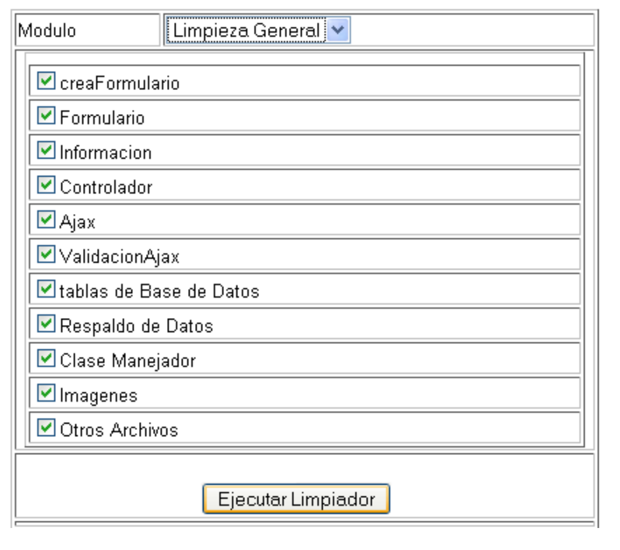


Se encuentra una lista con los módulos creados y al final una opción para Limpieza General.

En el caso de los módulos creados se carga una serie de opciones sobre lo que desea limpiar. Clic en el botón **Ejecutar Limpiador**

Automáticamente se elimina de todos lados un campo de referencia llamado dato.

Limpieza de Proyecto



En la opción **Limpieza General** se cargan otras opciones par la limpieza general del proyecto.

Se eliminan algunos archivos de la carpeta programador.

NOTA: Con la limpieza del proyecto no afecta el funcionamiento de las clases creadas. Solo se elimina la parte programadora de AplicaTem.

5. Generado de Reportes de AplicaTem

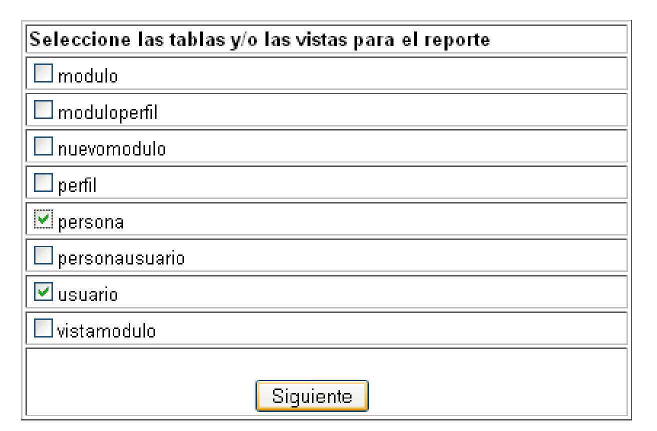
5.1 Generar Reportes.

Para generar reporte a través de AplicaTem debes iniciar sesión con una cuenta de administrador.

Clic en el modulo, Generar Reportes

La aplicación cargara el formulario correspondiente.

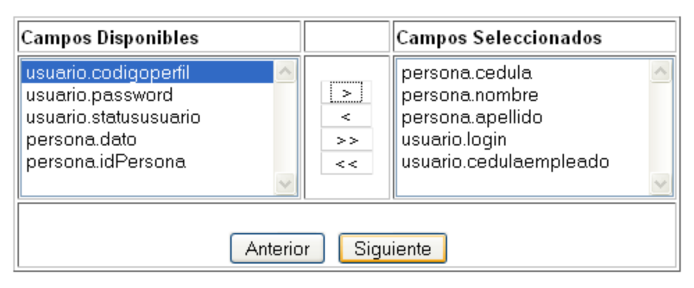
Selección de Tablas



Primero que nada debes seleccionar las tablas o las vistas las cuales van a intervenir en el reporte.

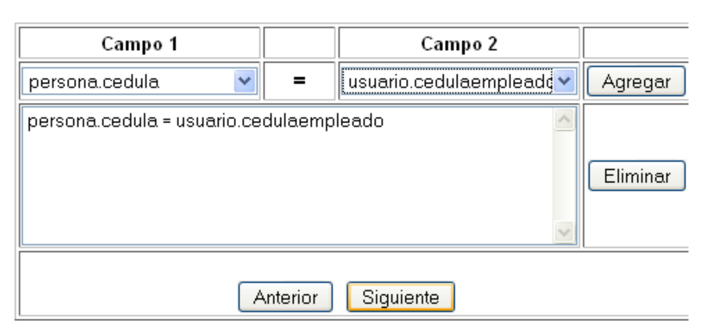
Clic en **Siguiente**

Seleccione los Campos para el Reporte

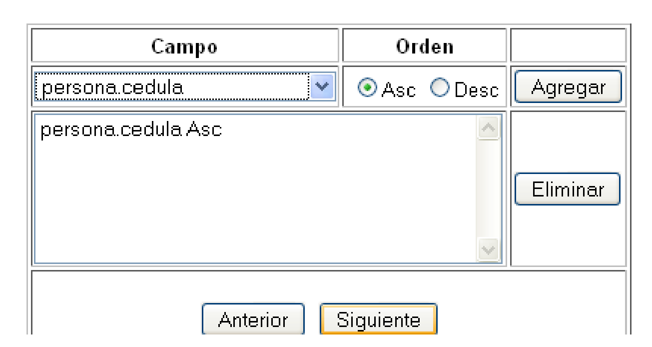


Se deben pasar los campos que fumaran parte de la consulta para el otro lado y luego dar clic en el botón **Siguiente**

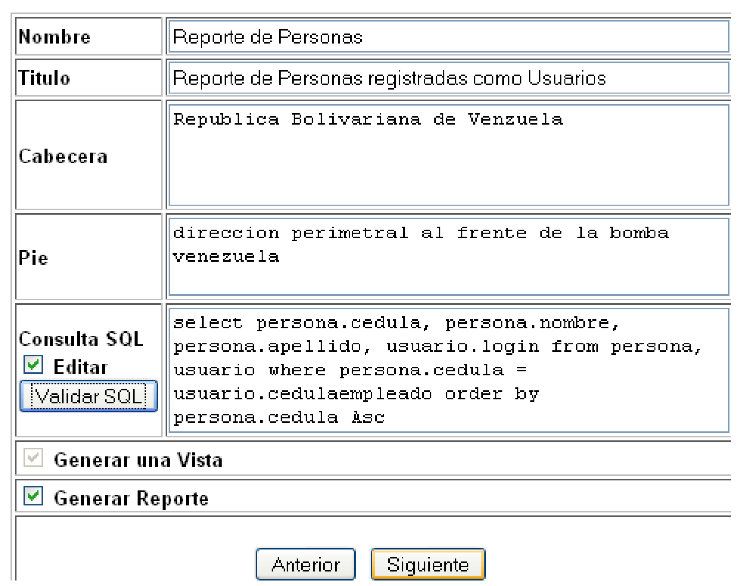
Seleccione los Campos a comparar



Este paso es opcional. Se deben agregar las condiciones necesarias para realizar la consulta o vista. Clic en **Siguiente**



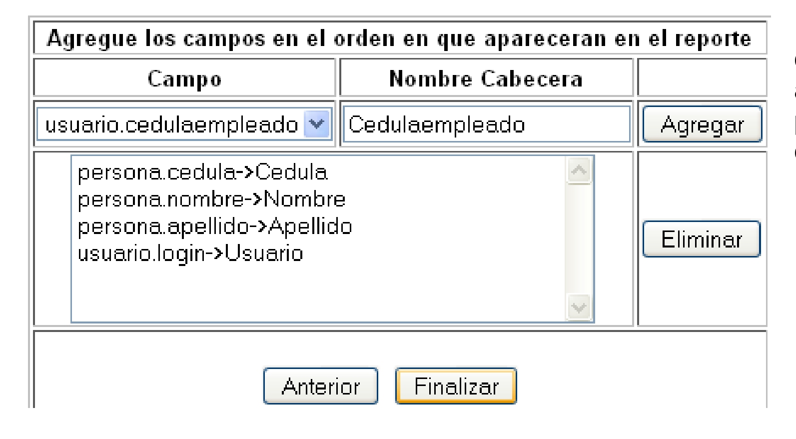
Este paso es Opcional. Para ir agregando el o los campos por los cuales se va a ordenar ya sea ascendente o descendente. **Clic en Siguiente.**



En este paso se configuran parámetros como el **nombre** que va a recibir el reporte, un **titulo,** dos parámetros opcionales como la **cabecera** y el **pie**.

Se puede validar la consulta para ver si no tiene errores y en caso de tener errores trae una opción para corregirlo automáticamente o también se puede corregir editando la consulta SQL.

Opcionalmente puedes generar una vista o reporte clic en **Siguiente**



En este último paso se agrega los campos en el mismo orden en que aparecerán en el reporte y también se puede modificar el nombre que mostrará en la cabecera. Clic en **Finalizar.**



A nivel de usuario.

Se creo un nuevo modulo llamado **Reporte de Personas**, donde se muestra un listado con los datos seleccionados en el generador de reportes.

Tiene una opción para imprimir el Reporte en PDF.



A nivel de Código.

- Se creo un archivo en **/Formulario/Rep_ReportedePersonas.php** donde se muestra el formulario para mostrar el reporte en formato HTML.
- Se creo un archivo en **/reportes/Rep_ReportedePersonas.php** donde se tiene el código para listar los datos en el datagrid.
- Se creo un archivo en **/reportes/Rep_ReportedePersonasImpreso.php** donde se encuentra el código PHP para generar el reporte en PDF.
- Se creo un nuevo registro de la tabla modulo y moduloperfil.
- Se creo una vista con el nombre de **vista_reportedepersonas** este solo se crea cuando intervienen varias tablas o vista en el reporte.

6. Proceso de carga de formularios

6.1 Cargar Formularios.

NOTA: Para cargar un formulario, éste debe existir en un archivo en la carpeta **Formulario/**

1. Clic en el enlace o botón [Iniciar Sesion] que se encuentra en index.html, este enlace internamente llama al método conexionPHP()

```
<a href="#" onclick="conexionPHP('formulario.php','Sesion')" >[Iniciar Sesion]</a>
```

2. Se encuentra en **/javascript/Ajax.js** es el único método en la aplicación que se conecta con el servidor a través de Ajax. Los dos parámetros que recibe representa el *archivo (formualario.php) al que va a entrar en el servidor* y el *case* (Sesion) donde va retornar lo que encuentre.

```
//para crear la conexion a traves de AJAX al servidor donde esta instalado php
//archivoPHP: nombre o ruta de archivo al que desea llamar;
//clase:  la clase con que desea comunicarse;
// cadena: la lista de parametros o datos adicionales;
//tipoDato:  la operacion que desea realizar)
function conexionPHP(archivoPHP,clase,cadena,tipoDato){
    //devuelve el numero de parametro recibidos
    var arg=conexionPHP.arguments.length;
    //crea el objeto AJAX
    var ajax=nuevoAjax();
    //abre la conexion con php
    ajax.open("POST", archivoPHP, true);
    ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    //envia los datos a traves de AJAX concatenados con =@
    if(archivoPHP=="administrar.php")
        ajax.send("d="+tipoDato+"=@"+clase+"=@"+cadena);
    else
        ajax.send("d="+clase+"=@"+cadena);

    ajax.onreadystatechange=function(){
        if (ajax.readyState==4){
            //obtiene la respuesta del servidor;  verifica la seguridad
            if(ajax.responseText=="SecurityFalse"){
                alert( "Error. Intento de Violación de Seguridad, la Sesion será reiniciada")
                conexionPHP('Seguridad/Seguridad.php','CerrarSesion');
            }
            else{
                if(clase=="DataGrid"){
                    document.getElementById(divDataGrid).innerHTML = ajax.responseText;
                }
                else{
                    //Hace la llamada a respuestaPHP() para que esla procesa la informacion devuelta
                    if(arg==4)
                        respuestaPHP(archivoPHP,clase,ajax.responseText,tipoDato);
                    else
                        respuestaPHP(archivoPHP,clase,ajax.responseText);
                }
            }
        }
    }
}
```

Figura Nº 10. Método conexionPHP

En la imagen anterior se puede observar como el método llama a un archivo PHP pasado en el primer parámetro por método POST, y se le envía la información a través de la variable **d** y concatenado con **=@** que es la notación de AplicaTem para concatenar cadenas.

6.2 Archivo formulario.php.

3. una vez en el archivo **/formulario.php**

```php
<?php
//Archivo que permite identificar que formulario se desea cargar
//verifica que antes se haya iniciado sesion
session_start();
$valor=split("=@",$_POST['d']);
$clase=$valor[0];
if($_SESSION["autenticacion"]!="On"){
    if($clase=='Sesion')
        include "Formulario/Sesion.php";
    else
        echo "SecurityFalse";
}
else{
    switch($clase)
    {
        case Sesion:
            //incluyo todo el formulario que se encuentre en ese archivo
            //se puede ejecutar codigo php en los archivos
            include "Formulario/Sesion.php";
            break;
        case Usuario:
            include "Formulario/Usuario.php";
            break;
        case Perfil:
            include "Formulario/Perfil.php";
            break;
        case Modulo:
            include "Formulario/Modulo.php";
            break;
    case Persona:
        include "Formulario/Persona.php";
        break;
    case NuevoModulo:
        include "Formulario/NuevoModulo.php";
        break;
            default:
                echo '<H3 align="center"><strong>Fo
    }// fin switch
}
?>
```

Separo la cadena que recibo en este caso hay un solo valor que es **d=Sesion=@**

Verifico si ha iniciado sesión, si no ha iniciado sesión compruebo si el primer parámetro es **Sesion** como es en este caso, cargo el formulario.

Si ha iniciado sesión ubico el case Sesion: y retorno todo lo que se consiga en el archivo ***include "/Formulario/Sesion.php"***

En este archivo se encuentra todo el formulario en código HTML sencillo e indentado para mayor entendimiento. Aunque también se puede interpretar código PHP.

4. Una vez que retorna el resultado a través de Ajax. Llama a la función

   ```
   respuestaPHP(archivoPHP,clase,ajax.responseText);
   ```

   Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (formulario.php) al cual fue, la clase o case (Sesion) a la cual entro y la respuesta que trajo del servidor.

5. En el método respuestaPHP()

   ```
   //funcion encargada de procesar toda la informacion devuelta del servidor.
   //archivoPHP: nombre o ruta de archivo con que establecio la conexion;
   //clase:  la clase con que desea comunicarse;
   // cadena: respuesta extraida del servidor
   //mensaje: algun mensaje adicional)
   function respuestaPHP(archivoPHP,clase,cadena,mensaje){
       //para limpiar la basura de la cadena
       cadena=limpiar(cadena);
       var arg=respuestaPHP.arguments.length;
       var capa=document.getElementById("principal");
   ```

   Captura el **div principal** en una variable capa

   Luego entra en el case "formuario.php"

   ```
   switch(archivoPHP)
   {
     case "formulario.php":
           claseGlobal=clase;
           //asigna la cadena al div principal
           capa.innerHTML=cadena;
   ```

   Y asigna la respuesta que trajo del servidor al div principal en formato HTML.

Introduzca su nombre de usuarios y contraseña

¡Bienvenido! Al introducir sus datos, el sistema iniciará una nueva sesión personalizada. De esta forma, Ud. podrá permitirá crear, modificar o eliminar la información relacionada con su cuenta.

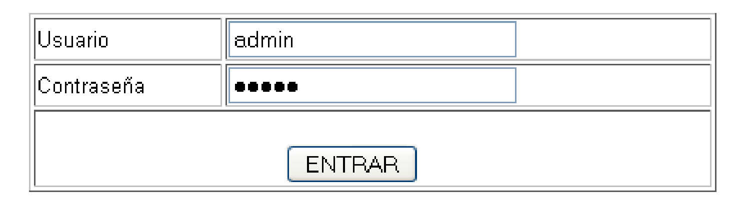


Figura Nº 14. Formulario de Iniciar Sesion.

7. Proceso de Restrinción de Usario

7.1 Proceso iniciar Sesion.

Una vez finalizado el proceso de cargar el formulario de iniciar sesión **(ver sección 5.1 Cargar Formularios)** y mostrado el formulario en el div principal **(ver Figura Nº 14. Formulario de Iniciar Sesion)**. Se procede completar los campos como nombre y contraseña, dar clic en el botón **ENTRAR** que contiene el enlace al método **iniciarSesion(),** en el archivo **/Formulario/Sesion.php**

```
<input  type="button" name="entrar" value="ENTRAR" onclick="iniciarSesion()">
```

En el método iniciarSesion(), que se encuentra en **/javascript/controlador.js**

```
function iniciarSesion(){
    loginUsuario=document.f1.login.value;
    conexionPHP("Seguridad/Seguridad.php","IniciarSesion",document.f1.login.value+"=@"+document.f1.password.value);
}
```

Se hace la llamada al método **conexionPHP(),** los dos parámetros que recibe representa el *archivo (**Seguridad.php**) al que va a entrar en el servidor*, el nombre del *case* (**IniciarSesion**) y el usuario concatenado con la contraseña.

El método conexionPHP **(ver Figura Nº 10. Método conexionPHP)** se encarga hacer la llamada a través de Ajax.

7.2 Archivo Seguridad.php.

Una vez en el archivo /**Seguridad/Seguridad.php**

```
<?php
//archivo que se encarga de administrar que funcios de seguridad desea ejecutar
session_start();
require_once("../DataBase/Acceso.php");
require_once "Usuario.php";
$valor=split("=@",$_POST['d']);
$clase=$valor[0];
$error='Error, el Usuario y/o Contraseña ingresados no son validos<br>';
$retorno=false;
switch($clase)
{
    case IniciarSesion:
        $objeto=new Usuario($valor[1],$valor[2],'','','');
        if(!$objeto->iniciarSesion($acceso)){
            echo "false";
        }
        break;
    case CerrarSesion:
        $objeto=new Usuario('','','','','');
        echo $objeto->cerrarSesion($acceso);
        break;
```

Entra en el **case IniciarSesion:** crea un objeto de la clase usuario, pasa como parámetro el nombre de usuario y la contraseña, y hace la llamada al método iniciarSesion de la clase Usuario.

Una vez en el método iniciarSesion() de la clase Usuario que se encuentra en **/Seguridad/Usuario.php**

```
public function iniciarSesion($acceso)
{
    //compruebo la veracidad de los datos del usuario
    $sql="select * from usuario
          where login='$this->login' and
          password='$this->password' and
          statususuario='Activo'";
    $acceso->objeto->ejecutarSql($sql);
    if($acceso->objeto->registros>0){
        $row=$acceso->objeto->devolverRegistro();
        //asigno perfil y cedula de la persona que inicia la sesion
        $this->codigoPerfil = trim($row['codigoperfil']);
        $this->codigoEmpleado = trim($row['codigoEmpleado']);
        //creo las variables de sesion
        $_SESSION["autenticacion"]='On';
        $_SESSION["permisologia"]=$permiso;
        $_SESSION["login"]= $this->login;
        return $this->cargarModulos($acceso);
    }
    else{
        return false;
    }
}
```

En este método se hace la consulta a la base de datos, verifica si el usuario esta registrado y Activo, asigna los datos a las variables, declara algunas variables de sesión y al final hace la llamada al método cargarModulos() que se encuentra en la misma clase

7.3 Proceso de cargar modulos.

```
function cargarModulos($acceso){
    $modulos="";
    $acceso->objeto->ejecutarSql("select * from vistamodulo where codigoperfil='$this->codigoPerfil' OR
    while($row=$acceso->objeto->devolverRegistro()){
        //esta condicion es propia de AplicaTem
        if(trim($row["namemodulo"])=='CreaFormulario' || trim($row["namemodulo"])=='LimpiarProyecto' ||
            $modulos=$modulos.'<li id="imagen"><a href="#" onclick="conexionPHP(\'Programador/creaFormu
        else
            $modulos=$modulos.'<li id="imagen"><a href="#" onclick="conexionPHP(\'formulario.php\',\''.
        $acceso->objeto->siguienteRegistro();
    }
    echo $modulos;
    return true;
}
```

Este método devuelve una cadena con todos los módulos que tiene asignado este perfil en el siguiente formato:

```
<li id="imagen"><a href="#" onclick="conexionPHP('formulario.php','NuevoModulo')">Nuevo Modulo</a></li>
```

Una vez que retorna el resultado a través de Ajax. Llama a la función

```
respuestaPHP(archivoPHP,clase,ajax.responseText);
```

Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (Seguridad/Seguridad.php) al cual fue, la clase o case (IniciarSesion) a la cual entro y la cadena concatenada que trajo del servidor. Luego entra en el **case "Seguridad/Seguridad.php"**

```
case "Seguridad/Seguridad.php":
    if(clase=='IniciarSesion'){
        if(cadena!="false"){
            document.getElementById("funcion").innerHTML=cadena;
            conexionPHP("validarExistencia.php","1=@personausuario","login=@"+loginUsuario,'
        }
        else
            alert( "Error, el Usuario y/o Contraseña ingresados no son validos. Por favor ir
    }
    else if(clase=='CerrarSesion'){
        if(cadena=="true")
            cerrarSesion();
    }
    else if(clase=='Sesion'){
        capa.innerHTML=cadena;
            iniciarSesion();
    }
break;
```

Entra en la condición de **IniciarSesion** si la cadena es diferente de false quiere decir que consiguió el usuario y asigna la cadena que trajo del servidor al **div funcion** que esta dentro del **div lateral**, luego hace la llamada para traer los datos del Usuario que esta entrando. Si la cadena es igual a false es porque el usuario o contraseña no existe y emite un mensaje de error.

8. Proceso de validación de datos

8.1 Validar datos

Para validar un campo de texto con uno de los tipos de datos ya definidos, es necesario hacer la llamada desde javascript como el ejemplo que se muestra a continuación:

```
if(validaCampo(document.f1.cedula,isCedula)){}
```

Donde **validaCampo()** es el nombre de la función en javascript que se encarga de validar todos los tipos de datos, esta función recibe 4 parámetros; el campo que va a validar sin el valúe **document.f1.cedula**, la forma en que lo va a validar en este caso es como **isCedula**.

```
if(validaCampo(document.f1.cedula,isCedula,true,'Error, cedula incorrecta')){}
```

Los dos para metros opcionales, son un valor true o false, por defecto **false** si va a recibir datos en blanco y el último parámetro un mensaje de error personalizado, por defecto ya trae definido un mensaje de error para cada uno de los tipos de validaciones.

Retorna true en caso de validar con éxito el campo e imprime el mensaje de error y retorna false en caso de ser datos erróneos.

8.2 Archivo validacion.js

Función **validaCampo()** ubicada en **/javascript/ validacion.js**

```
function validaCampo(campo, funcion, vacio, mensaje)
{
    var msg;
    if (validaCampo.arguments.length < 3) vacio = defaultEmptyOK;
    if (validaCampo.arguments.length == 4) {
        msg = mensaje;
    } else {
        if( funcion == isAlphabetic) msg = pAlphabetic;
        if( funcion == isInteger) msg = pInteger;
        if( funcion == isAlphanumeric) msg = pAlphanumeric;
        if( funcion == isNumber ) msg = pNumber;
        if( funcion == isEmail ) msg = pEmail;
        if( funcion == isPhoneNumber ) msg = pPhoneNumber;
        if( funcion == isName ) msg = pName;
        if( funcion == isCedula ) msg = pCedula;
        if( funcion == isPassword ) msg = pPassword;
        if( funcion == isTexto ) msg = pTexto;
        if( funcion == isSelect ) msg = pSelect;
        if( funcion == isDate ) msg = pDate;
    }
    if ((campo.value == null) || (campo.value.length == 0)){
        alert(mMessage);    campo.select();
       return false;
    }
    if (funcion(campo.value) == true){  return true;}
    else{
        alert(pPrompt + msg); campo.select(); return false;
    }
}
```

Primero verifica la cantidad de parámetros pasadas si es menor de tres quiere decir que no personalizó el tercer parámetro y asigna el valor por defecto que es false.

Luego verifica si se paso el cuarto parámetro para personalizar el mensaje de error y asigna la cadena pasada, si no se pasa el mensaje personalizado el verifica el segundo parámetro para ver el tipo de validación y dependiendo de éste asigna el mensaje que tiene por defecto para esa función.

Verifica el campo, si esta vacío emite mensaje de error para cuando el campo este vacío y retorna false si no esta vacío hace el llamado a la función pasada por parámetro que puede ser cualquiera de las listadas en esta función, si la función retorna false se imprime el mensaje de error y se devuelve false.

NOTA: Las funciones para validar los diferentes tipos de datos se encuentran definidas e implementadas en este mismo archivo

Listado de mensajes de error por defectos

```
var pPrompt = "Error: ";
var mMessage = "Error: no puede dejar este espacio vacio"
var pAlphanumeric = "ingrese un texto que contenga solo letras y/o numeros";
var pAlphabetic   = "ingrese un texto que contenga solo letras";
var pTexto    = "ingrese un texto que contenga solo letras numeros y \"\'.,-;";
var pInteger = "ingrese un numero entero";
var pNumber = "ingrese un numero";
var pPhoneNumber = "ingrese un número de teléfono de 11 digitos";
var pEmail = "ingrese una dirección de correo electrónico válida";
var pName = "ingrese un texto que contenga solo letras y/o espacios";
var pCedula = "ingrese un numero de cedula de 8 digitos";
var pPassword = "la contraseña debe ser minimo 6 digitos";
var pSelect = "no ha Seleccionado ninguna opcion en el Select";
var pDate = "debe introducir una fecha en este formato  DD/MM/AAAA";
```

Estos mensajes por defectos se pueden encontrar al inicio del archivo **/javascript/ validacion.js**

9. Proceso de Administración de clases

9.1 Administrar clases

Para administrar (incluir, modificar o eliminar) una clase o tabla, primero que nada se debe cargar el formulario de la clase correspondiente, introducir los datos en el formulario y teclear el botón correspondiente dependiendo de la operación que desea registrar.

Los botones internamente tienen definido la llamada al método **verificar()**:

```
<input  type="button" name="registrar" value="REGISTRAR" onclick="verificar('incluir','NuevoModulo')">
<input  type="button" name="modificar" value="MODIFICAR" onclick="verificar('modificar','NuevoModulo')">
<input  type="button" name="eliminar" value="ELIMINAR" onclick="verificar('eliminar','NuevoModulo')">
<input  type="reset" name="Resetear" value="CANCELAR">
```

Esto se puede observar en el formulario de la clase que desea administrar, en este caso estamos tomando como ejemplo la clase NuevoModulo, que se encuentra en **/Formulario/NuevoModulo.php,** se puede observar que se hace la llamada a la función verificar y se le pasan dos parámetros, el primero es la **operación** (incluir, modificar o eliminar) que se desea realizar y el segundo es la **clase** que se desea administrar.

Una vez en la función **verificar()** que se encuentra en **/javascript/controlador.js**

```
//es llamada cuando se desea incluir, modificar o eliminar datos de una tabla.
//tipoDato: representa la operacion que desea hacer: incluir, modificar o eliminar
//clase: a que  clase o tabla  desea hacer la operacion
function verificar(tipoDato,clase)
{
   switch(clase)
   {
     //clase o tabla usuario
     case "Usuario":
```

Esta función internamente esta basada en **switch** o **case**. Los case representan las clases por ello se debe ubicar el **case "NuevoModulo",** que es la clase que estamos tomando como ejemplo.

```
case "NuevoModulo":
    if(validaCampo(document.f1.cedula,isCedula) && validaCampo(document.f1.nombre,isName) &&
    {
        if(confirmacion(tipoDato,clase,cedula()+"=@"+nombre()+"=@"+estadoCivil()+"=@"+verRadi
            document.f1.reset();
    }
    break;
```

En este case primero que nada se validan los campos uno por uno para comprobar que estén introducidos correctamente, en caso de estar un dato erróneo se muestra un mensaje de error y vuelve al formulario. Al validar todos los campos se procede a llamar al método confirmación y se le envía la **operación**, la **clase** y la **lista de parámetros** para mostrar la confirmación

```
function confirmacion(tipoDato,clase,cadena){
    if (confirm('¿seguro que desea enviar este formulario?')){
        //hace la llamada para hacer la conexion con php
        conexionPHP("administrar.php",clase,cadena,tipoDato);
        return true;
    }
    else
        return false;
}
```

En el método confirmacion() lo que se hace es confirmar el envío de los datos al servidor y se hace el llamado al método **conexionPHP (ver Figura Nº 10. Método conexionPHP)**, que es el método para comunicarse con el servidor a través de Ajax y se le envía el nombre del archivo (**administrar.php**) donde va a entrar, la **clase**, la **lista de parámetros** y la **operación** a realizar

9.2 Archivo administrar.php

Este archivo se encuentra ubicado en la carpeta principal **/administrar.php**

Administrar.php, es un archivo que se encarga de crear el o los objeto de una o varias clases y hacer las llamadas correspondientes a los métodos de esas clases dependiendo de los parámetros recibidos.

El archivo consta de 3 partes:

```
<?php
//ARCHIVO ENCARGADO DE ADMINISTRAR QUE SE QUIERE HACER Y A QUE CLASE
//ES DECIR 'incluir', 'modificar' o 'eliminar' a la clase 'Persona'
session_start();
if($_SESSION["autenticacion"]!="On"){
    echo "SecurityFalse";
}
else{
require_once("DataBase/Acceso.php");
$acceso=conexion();
//recibe los datos enviados
$cadena=$_POST['d'];

//divide la cadena recibida cada vez que consiga '-Class-', esto quiere decir que se va a hac
$clases=split("-Class-",$cadena);
$mensaje=false;
//cuenta la cantidad de clases
$x=count($clases);

$y=0;
```

Se verifica el inicio sesión, luego se importa el archivo para la conexión con la base de datos.

$cadena almacena toda la cadena recibida como por ejemplo **Incluir=@NuevoModulo=@19447744=@Jesus=@Soltero=@Masculino=@1987-05-28**

Se divide la cadena en **-Class-** cada vez que aparece esta palabra en los parámetros recibidos quiere decir que se va a crear un nuevo objeto para una nueva clase y recibe otra cadena como la de ejemplo.

Por ultimo se inicializan algunas variables, entre ellas **$x** que representa la cantidad de clases a crear objetos.

```php
//recorre el ciclo dependiendo de la cantidad de clases que se desea instaciar
for($y=0;$y<$x;$y++){
    //divide la cadena de la primeta clase en '=@'
    $valor=split("=@",$clases[$y]);
    //la posicion '$valor[0], representa la operacion de desea realizar ya sea incluir modificar o eliminar
    $boton=$valor[0];
    //la posicion '$valor[0], representa la clase.
    $nombreClase=$valor[1];
    if($nombreClase=="Modulo" || $nombreClase=="ModuloPerfil" || $nombreC
        require_once "Seguridad/".$nombreClase.".php";
    else
        require_once "Clases/".$nombreClase.".php";

    if(class_exists($nombreClase)){
        //se crea el objeto de la clase
        $objeto = llamada($nombreClase,$valor);
        //concateno la funcion a la que voy a llamar ejemplo 'incluirPersona'
        $metodo=$boton.$nombreClase;
        //verifico la existencia del campo clave
        if(!$objeto->validaExistencia($acceso)){
            //verifica si vas a incluir
            if($boton=="incluir" || $nombreClase=="ModuloPerfil"){
                //haces la llamada al metodo
                $objeto->$metodo($acceso);
                $mensaje=true;
            }
            else{
                echo ', EL CAMPO NO ESTA REGISTRADO EN ESTA TABLA';
                $mensaje=false;
            }
        }
        else{
            if($boton!="incluir" || $nombreClase=="ModuloPerfil"){
                $objeto->$metodo($acceso);
                $mensaje=true;
            }
            else{
                echo ', EL CAMPO YA ESTA REGISTRADO EN ESTA TABLA';
                $mensaje=false;
            }
        }
    }
    else echo 'LA CLASE NO EXISTE';
    if(!$mensaje){ break;}
}
```

*El ciclo se repite la cantidad de clases que ha conseguido.

*Se divide la primera clase en **=@**

*En el ejemplo:
$boton="incluir";
$nombreClase="NuevoModulo";

*Se hacen unas comparaciones para importar las clases.

*Se verifica si la clase existe

*Se crea el objeto de la clase a través del método **llamada()**

*se construye el método con la combinación de las variable
$metodo="incluirNuevoModulo";

*se valida la existencia del campo clave, en caso de que no exista en la base de datos solo se puede incluir.

*Finalmente se procede a llamar al método incluirNuevoModulo(); y se le pasa por parámetro el objeto de conexión con la base de datos

```php
function llamada($nombreClase,$args)
{
    //verifico la catidad de parametros
    switch (count($args))
    {
        Case 4:
            //ejemplo en caso de ser 4 parametros el primeto es la 'operacio' el segundo la 'clase' y los demas atributos de la clase
            $objeto=new $nombreClase($args[2],$args[3]);break;
        case 5:
            $objeto=new $nombreClase($args[2],$args[3],$args[4]);break;
        case 6:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5]);break;
        case 7:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6]);break;
        case 8:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7]);break;
        case 9:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
        case 10:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
        case 11:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
        case 12:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
        case 13:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
        case 14:
            $objeto=new $nombreClase($args[2],$args[3],$args[4],$args[5],$args[6],$args[7],$args[
```

La ultima parte de este archivo es la creación del objeto a través de un método que lleva por nombre **llamada()** y recibe dos parámetros el **nombre de la clase** y la **lista de atributos de la clase** sin incluir las dos primeras posiciones

El switch recibe la cantidad de argumentos que tiene el arreglo y se crea un case para cada ocasión dependiendo de los argumentos recibidos

En el ejemplo recibe 7 argumentos
Los dos primeros representan la operación y la clase y los otros 5 los atributos de la clase por lo tanto entra en el case 7 que la clase recibe 5 parámetros. En el ejemplo:
$objeto=new NuevoModulo("19447744","Jesus","Soltero","Masculino","1987-05-28");

Al finalizar el método retorna el objeto de la clase.

En el método **incluirNuevoModulo()** esta definido en la clase NuevoModulo que se encuentra en **/ClasesNuevoModulo.php**

```
public function incluirNuevoModulo($acceso){
    return $acceso->objeto->ejecutarSql("insert into NuevoModulo(cedula,nombre,estadoCivil,sexo,fechaNac,dato) values
    ('$this->cedula','$this->nombre','$this->estadoCivil','$this->sexo','$this->fechaNac','$this->dato')");
}
```

Al registrarse todo con éxito el archivo retorna la cadena **"true";**

Una vez que retorna el resultado a través de Ajax, se hace la llama desde el método **conexionPHP** a la función

```
respuestaPHP(archivoPHP,clase,ajax.responseText);
```

Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (administrar.php) al cual fue, la clase(NuevoModulo) a la cual instancio y la cadena concatenada que trajo del servidor. Luego entra en el **case "administrar.php"** del método respuestaPHP

```
case "administrar.php":
  if(clase!="ModuloPerfil"){
      if (cadena=="true"){
            alert( "TRANSACCIÓN COMPLETADA CON EXITO ");
            if(clase=="NuevoMundo" || clase=="Modulo" || clase=="Perfil" || clase=="Persona"
                conexionPHP('formulario.php',clase);
            }
      }
      else{
          alert( "ERROR DURANTE TRANSACCIÓN: "+cadena);
      }
  }
break;
```

Si la cadena, que es el resultado traído del servidor, es igual a "**true"**, es porque la operación se realizó con éxito e imprime el mensaje de confirmación.

Administracion de NuevoModulo



Luego verifica si la clase se encuentra en la condición, hace la llamada para cargar el formulario de esa clase

Si cadena es diferente de **"true"** se imprime el mensaje de error y se imprime la cadena para saber porque no se registro los datos.

10. Proceso Validación de Existencia de Datos

10.1 Validar existencia de un campo.

Para validar la existencia de un dato en la base de datos, primero que nada se debe cargar el formulario de la clase o tabla correspondiente,

El campo a validar debe tener asociado un evento y en el hacer la llamada a una función como ejemplo

```
<input  type="text" name="cedula" maxlength="8" size="30"onChange="validarNuevoModulo()" value="">
```

Por lo general la validación de existencia de datos se le aplica a los campos claves de las tablas como es en este caso y los métodos llamados están definidos en **/javascript/controlador.js**

```
function validarNuevoModulo(){ conexionPHP("validarExistencia.php","1=@nuevomodulo","cedula=@"+cedula());}
```

Como se puede observar este método **validarNuevoModulo()** hace la llamada al metodo **conexionPHP (ver Figura Nº 10. Método conexionPHP)**, y le envía el nombre del archivo (**validaExistencia.php**) donde va a entrar, de segundo parámetro le envía un **1** que representa el tipo de operación, concatenado con el nombre de la tabla **NuevoModulo** a la cual se le va a consultar y por ultimo una cadena con la palabra **cedula** concatenada con el valor del campo **"19447744"**.

10.2 Archivo validarExistencia.php

```
require_once("DataBase/Acceso.php");
$acceso=conexion();
$cadena=$_POST['d'];
//recibe una cadena concatenada con '=@', separa la cadena
$palabras=split("=@",$cadena);
//el primer valor representa el tipo de consulta.
$consulta=$palabras[0];
//el segundo valor la tabla que se desea consultar
$tabla=$palabras[1];
if($consulta=="0"){
    $select="select * from ".$tabla;
}
else{
    //los otros dos valores representan el dato que deseas comparar y el valor para compararlo
    $dato=$palabras[2];
    $codigo=$palabras[3];
}
if($consulta=="1"){
    //crea la consulta SQL
    $select="select * from ".$tabla." where ".$dato."=".'".$codigo."'";
}
else if($consulta=="2"){
    //en caso de que sea una consulta a una tabla con dos comparaciones
    $dato1=$palabras[4];
    $codigo1=$palabras[5];
    $select="select * from ".$tabla." where ".$dato."=".'".$codigo."'"." and ".
}
//ejecuto la sentencia SQL
$acceso->objeto->ejecutarSql($select);
//devuleve los valores
if($row=$acceso->objeto->devolverRegistro()){
    $num = count($row);
    //imprime la cantidad de valores
    echo $num/2;
    for($i=0;$i<$num;$i++){
        if($row[$i]=="")
            break;
        //va concatenando cada valor con '=@'
        echo '=@'.$row[$i];
    }
}
else{
    //si no existe retorna la cadena 'false'
    echo 'false';
}
```

Este archivo se encuentra ubicado en la carpeta principal **/validarExistencia.php**

validarExistencia.php, es un archivo que se encarga de hacer consultas a la base de datos para traer los datos de un registro en caso de que existan.

*Se verifica el inicio sesión,

*se crea la conexión con la base de datos.

***$cadena** almacena toda la cadena recibida como por ejemplo **1=@nuevomodulo=@cedula=@19447744**

*se verifica el tipo de consulta para almar el SQL, en el ejemplo tomado entra en la condición consulta igual a 1

*Se procede a crear el SQL, quedaría **$select=" select *from nuevomodulo where cedula='19447744' ";**

***se ejecuta la consulta y se procede a verificar si encontró algún registro, en este caso procede a retornar todo el registro concatenado con **=@** ejemplo.
5=@19447744=@Jesus=@Soltero=@Masculino=@1987-05-28

El numero 5 de primero representa el numero de columnas retornadas.

En caso de no existir se retorna la cadena **"false".**

Una vez que retorna el resultado a través de Ajax, se hace la llama desde el método **conexionPHP** a la función `respuestaPHP(archivoPHP,clase,ajax.responseText);`

Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (validarExistencia.php) al cual fue, la tabla (**nuevomodulo**) a la cual consulto y la cadena concatenada que trajo del servidor. Luego entra en el **case "validarExistencia.php"** del método respuestaPHP.

```
case "validarExistencia.php":
    if (cadena!="false"){
        var tabla=clase.split("=@")
        asignarCampos(tabla[1],cadena);
        boton(true,false,false,claseGlobal);

    }
    else if(cadena=="false"){
        boton(false,true,true,claseGlobal);
    }
break;
```

En este case se comprueba si la cadena es diferente de **"false"** quiere decir que encontró un registro.

Se procede a separa la clase que es igual a "**1=@nuevomodulo**" y se hace la llamada al método **asignarCampos()** y se le envía por parámetro la tabla y la cadena con los datos del registro.

Luego de la llamada al metodo se procede a activar y desactivar los botones

```
//permite asignar los campos traidos de una tabla.
//tabla: la tabla de donde fueron extraidos los datos
//cade: una cadena con todos los datos concatenados con =@
function asignarCampos(tabla,cade)
{
    //divide los datos en un arreglo
    cadena= cade.split("=@");
    var i=0;
    for(i=0;i<cadena.length;i++)
    {
        //limpia cada uno de los datos
        cadena[i]=trim(cadena[i]);
    }
    switch(tabla)
    {
        case "usuario":
```

Este método se encuentra en **/javascript/validacionAjax.js**

*se separa la cadena concatenada en un arreglo.

*se limpia los espacios en blanco a cada uno de los datos.

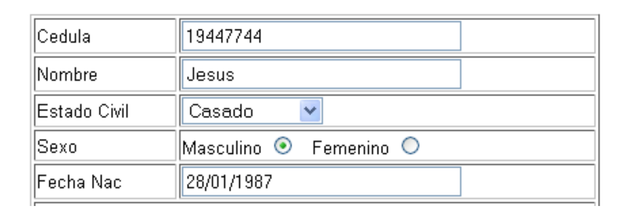
*Para cada tabla se crea un case con el nombre de esa tabla.

```
case "nuevomodulo":
    document.f1.dato.value=cadena[1];
    document.f1.cedula.value=cadena[2];
    document.f1.nombre.value=cadena[3];
    document.f1.estadoCivil.value=cadena[4];
    traeRadiosexo(cadena[5]);
    document.f1.fechaNac.value=formatdatei(cadena[6]);
    break;
```

Una vez en el **case “nuevomodulo”** se procede a signar cada uno de los datos traído de la base de datos a su respectivo campo.

Al final aparecerá como se muestra el la siguiente imagen

Administracion de NuevoModulo



11. Proceso de carga de Información

11.1 Cargar Información.

Para realizar un proceso de carga de información es necesario hacer la llamada a un método. Como no tenemos mucha información que cargar vamos a tomar como ejemplo una consulta a la base de datos e imprimimos el nombre.

El campo cedula vamos a asociarle un evento que va a llamar a una función **cargaInfo()**

```
<input  type="text" name="cedula" maxlength="8" size="30"onChange="cargaInfo()" value="">
```

Por lo general los métodos para carga de información se desarrollan al final de **/controlador.js**

```
function cargaInfo(){
    conexionPHP("informacion.php","cargaInforma",cedula());
}
```

Como se puede observar este método **cargaInfo()** hace la llamada al metodo **conexionPHP (ver Figura Nº 10. Método conexionPHP)**, y le envía el nombre del archivo (**informacion.php**) donde va a entrar, una cadena "**cargaInforma"** que representa el case donde va a entrar, y un parámetro adicional como la cedula introducida en el campo **cedula**

11.2 Archivo nformacion.php

Este archivo se encuentra ubicado en la carpeta principal **/informacion.php**

Informacion.php, es un archivo que se encarga de realizar la llamada a algunos métodos para realizar algunos procesos en PHP y retornar información.

```
<?php
//archivo destinado a procesar y devolver datos o informacion relaciona con la a
session_start();
require_once "procesos.php";
$cadena=$_POST['d'];
$valor=split("=@",$cadena);
$clase=$valor[0];

if($_SESSION["autenticacion"]!="On"){

    if($clase=='Manejador')
        echo Manejador();
    else
        echo "SecurityFalse";
}
else{
    switch($clase)
    {
        case cargaInforma:
            echo  carga_informa($acceso,$valor[1]);
        break;
```

*Se verifica el inicio sesión,

***$cadena** almacena toda la cadena recibida como por ejemplo **cargaInforma=@19447744**

*se crea un case con el nombre del primer parámetro **case cargaInforma:**

*Dentro del case se hace la llamada al método **carga_informa()** y se le envia la conexión con la base de datos y el parámetro adicional la **cedula** para que procese la información .

```
function carga_informa($acceso,$cedula){
    $acceso->objeto->ejecutarSql("select *from nuevomodulo where cedula='$cedula'");
    $row=$acceso->objeto->devolverRegistro();
    return trim($row["nombre"]);
}
```

En el método **carga_informa()** proceso la información y retorno el nombre al que corresponde la cedula.

Una vez que retorna el resultado a través de Ajax, se hace la llama desde el método **conexionPHP()** a la función `respuestaPHP(archivoPHP,clase,ajax.responseText);`

Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (informacion.php) al cual fue, el case (**cargaInforma**) al cual entro y la cadena con el nombre que trajo del servidor. Luego entra en el **case "informacion.php"** del método respuestaPHP

```
case "informacion.php":
  if(clase=="cargaInforma"){
      alert(cadena);
  }
```

Aquí comprueba si es el case en donde el entro en información y muestra la cadena en un mensaje.



12. Proceso de Visualizar Reportes

12.1 Visualizar Reportes en HTML.

Para visualizar un reporte en HTML, se debe cargar el formulario del reporte correspondiente, este proceso de carga de formulario del reporte es similar a la carga de un formulario común, también existe un archivo en la carpeta **/Formulario/** llamado **"Rep_" mas el nombre que se le da cuando se crea.**

Para este ejemplo vamos a tomar como ejemplo el reporte que se creo en el capitulo **5.1 Generar Reportes** de este manual.

El archivo se encuentra ubicado en **/Formulario/Rep_ReportedePersonas.php**

```
<br>
<H3>Reporte de Personas registradas como Usuarios</H3>
<br>
<form name="f1">
    <table border="0" width="100%" align="CENTER">
        <tr>
            <td colspan="2">
                <div id="datagrid">

                </div>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input  type="button" value="IMPRIMIR REPORTE" onclick="ImprimirRep_ReportedePersonas()" n
            </td>
        </tr>
    </table>
</form>
```

Dentro del formulario se creo un div llamado **datagrid** que es donde se va a cargar el listado del reporte.

Una vez que retorna el resultado a través de Ajax. Llama a la función

```
respuestaPHP(archivoPHP,clase,ajax.responseText);
```

Que se encuentra en **/javascript/Ajax.js** y se le envía el archivo (formulario.php) al cual fue, el case (**Rep_ReportedePersonas**) a la cual entro y el formulario que trajo del servidor.

Captura el **div principal** en una variable capa

Luego entra en el case "formuario.php"

```
case "formulario.php":
    claseGlobal=clase;
    //asigna la cadena al div principal
    capa.innerHTML=cadena;
```

Y asigna la respuesta que trajo del servidor al div principal en formato HTML.

```
if(clase=="Rep_ReportedePersonas"){
    archivoDataGrid="reportes/Rep_ReportedePersonas.php";
    updateTable();
}
```

Luego en el mismo case consigue una condición y hace la llamada a la carpeta /**reporte/Rep_ReportedePersonas.php** desde el método **updateTable()**

Este método se encuentra ubicado en la carpeta propia de la **librería eyedatagrid /include/ eyedatagrid/ eyedatagrid.js**

```
var archivoDataGrid="";
var claseDataGrid="DataGrid";
var divDataGrid="datagrid";
function updateTable() { conexionPHP(archivoDataGrid,claseDataGrid,divDataGrid);}
```

```
<?php
require_once("../DataBase/Acceso.php");
require '../include/eyedatagrid/class.eyemysqladap.inc.php';
require '../include/eyedatagrid/class.eyedatagrid.inc.php';
$x = new EyeDataGrid($db);

$x->setQuery("cedula,nombre,apellido,login","vista_reportedepersonas","","");
$x->setColumnHeader("cedula", "Cedula");
$x->setColumnHeader("nombre", "Nombre");
$x->setColumnHeader("apellido", "Apellido");
$x->setColumnHeader("login", "Usuario");

$x->hideOrder();
$x->showRowNumber();
$x->setResultsPerPage(20);
$x->printTable();
?>
```

En este archivo se importa la librería para la creación de datagrid

La llamada al método **setQuery()** recibe como primer parámetro una lista de los campos a mostrar, el nombre de la tabla o vista, opcionalmente puedes pasarle el campo clave y una condición.

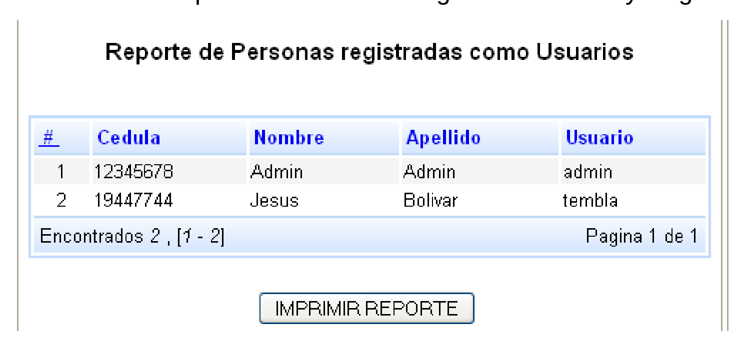
Luego se le cambia el nombre a las cabeceras pasándole dos parámetros al método **setColumnHeader()** el nombre de la columna en la tabla y el nombre por el cual vas a reemplazar ese nombre, esta paso es opcional ya que si no se cambia el nombre a la cabecera se coloca el mismo nombre que tiene.

Y por ultimo llamada a funciones como ocultar orden, mostrar el número de columnas, imprimir máximo 20 resultados por página y la impresión de la tabla.

Una vez que retorna el resultado a través de Ajax. En el método **conexionPHP()**

```
if(clase=="DataGrid"){
    document.getElementById(divDataGrid).innerHTML = ajax.responseText;
}
else{
    //Hace la llamada a respuestaPHP() para que esla procesa la informacion devuelta
    if(arg==4)
        respuestaPHP(archivoPHP,clase,ajax.responseText,tipoDato);
    else
        respuestaPHP(archivoPHP,clase,ajax.responseText);
}
```

Se hace la comparación si clase es igual a DataGrid y asigna la respuesta que trajo al div datagrid.



Reporte de Personas registradas como Usuarios

#	Cedula	Nombre	Apellido	Usuario
1	12345678	Admin	Admin	admin
2	19447744	Jesus	Bolivar	tembla

Encontrados 2 , [1 - 2] Pagina 1 de 1

IMPRIMIR REPORTE

12.2 Visualizar Reportes en PDF.

Para visualizar un reporte en PDF, se debe mostrar el reporte en formato HTML, este reporte tiene un botón para **IMPRIMIR REPORTE** que hace la llamada a al método

```
function ImprimirRep_ReportedePersonas(){
    location.href="reportes/Rep_ReportedePersonasImpreso.php";
}
```

Se hace de esta manera a través de un método para poder facilitar la creación de reporte específicos pasándole parámetros opcionales.

Se hace un link con el archivo **/reportes/Rep_ReportedePersonasImpreso.php**

```
<?php
require('../include/FPDF/fpdf.php');
require_once "../procesos.php";

class PDF extends FPDF
{
```

En el archivo se incluye la librería **FPDF** que es para la creación de páginas en formato PDF, y la librería proceso que ya contiene la conexión con la base de datos.

Se crea una nueva clase llamada PDF que hereda de la clase FPDF

*Se crea el objeto de la clase PDF

```
//crea el objeto pdf
$pdf=new PDF();
//salto de página automático desde la parte inferior de la página
$pdf->SetAutoPageBreak(true,35);
//agrega una nueva pagina
$pdf->AddPage();
$pdf->Fecha();
$pdf->Titulo();
$pdf->Cuerpo($acceso);
//imprime el reporte en formato PDF
$pdf->Output('reporte.pdf','D');
?>
```

*Se le dice que salte 35 puntos al final de cada página

*Agrego una nueva página, automáticamente aquí hace la llamada al método **Header()** y **Footer(),** que es para mostrar una cabecera a la pagina y el pie, son llamados en cada página.

*Se la llamada a varios métodos como **Fecha()**, **Titulo()**, **Cuerpo()**, y por ultimo al método **Output(),** que genera la pagina PDF

```
//Cabecera del Reporte aparecera en todas las paginas
function Header()
{
    //$this->Image('../imagenes/cabecera.jpg',20,10,40);
    $this->SetFont('Arial','B',10);
    $this->SetXY(15,18);
    $this->MultiCell(180,5,utf8_decode('REPUBLICA BOLIVARIANA DE VENZUELA'),'0','C');
    $this->Ln();
}
```

El metodo **header()** es llamado automáticamente en cada pagina y lo que hace es imprimir la cadena en Arial 10, negrita.

```
function Footer()
{
    //$this->Image('../imagenes/pie.jpg',15,250,170);
    $this->AliasNbPages();
    $this->SetY(-30);

    $this->SetFont('Arial','B',9);
    $this->MultiCell(180,5,utf8_decode('direccion perimetral al frente de la bomba venezuela'),'0','C');

    $this->SetFont('Arial','I',8);
    $this->Cell(0,7,'Pag. '.$this->PageNo().' / {nb}',0,1,'C');

}
```

El método **Footer()** imprime una cadena en el pie de cada página y también muestra el numero de pagina actual.

```
//para mostrar las fecha de impresion del reporte
function Fecha()
{
    $this->SetFont('Arial','B',8);
    $this->SetX(133);
    $this->Cell(12,5,'Fecha:',0,0,'L');
    $this->SetFont('Arial','',8);
    $this->Cell(18,5,date("d/m/Y"),0,0,'L');
    $this->SetFont('Arial','B',8);
    $this->Cell(12,5,'Hora:',0,0,'L');
    $this->SetFont('Arial','',8);
    $this->Cell(18,5,date("h:i:s A"),0,0,'L');
    $this->Ln();
}
```

El método **Fecha()** lo que hace es mostrar la fecha en la primera pagina.

```
//Titulo del reporte
function Titulo()
{
    $this->SetFont('Arial','B',10);
    $this->SetTextColor(0,0,0);
    $this->MultiCell(190,7,utf8_decode('REPORTE DE PERSONAS REGISTRADAS COMO USUARIOS'),'0','C');
    $this->Ln();
}
```

El método **Titulo()** muestra un titulo en la primera pagina del reporte.

```
//muestra los titulos de los campos de los reportes
function TituloCampos()
{
    $this->SetFillColor(244,249,255);
    $this->SetDrawColor(225,240,255);
    $this->SetLineWidth(.2);
    //dimenciones de cada campo
    $w=array(10,42,42,42,44);
    $header=array('Nro','Cedula','Nombre','Apellido','Usuario');
    $this->SetFont('Arial','B',9);
    $this->SetX(15);
    for($k=0;$k<count($header);$k++)
        $this->Cell($w[$k],7,$header[$k],1,0,'J',1);
    $this->Ln();
    return $w;
}
```

El método **TituloCampo()** es llamado desde el método **Cuerpo()** para mostrar la cabecera de la columna y asignar el tamaño que va a tener cada columna del reporte.

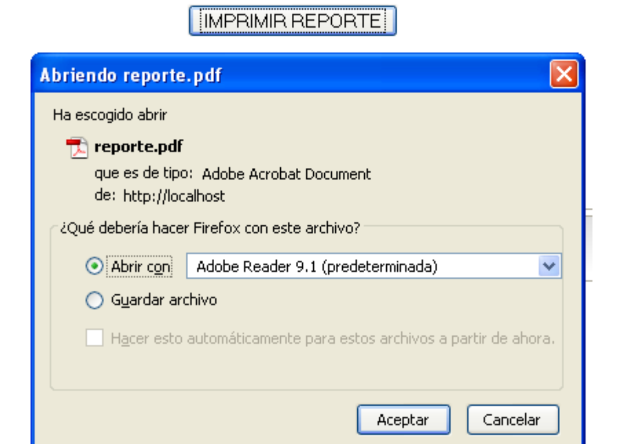
```
}
//muestra el listado de los reportes
function Cuerpo($acceso)
{
    $w=$this->TituloCampos();
    $acceso->objeto->ejecutarSql("SELECT *FROM vista_reportedepersonas");

    $this->SetFont('Arial','',9);
    $cont=1;
    $this->SetFillColor(249,249,249);
    $this->SetTextColor(0);
    while ($row=row($acceso))
    {
        $this->SetX(15);
        $this->Cell($w[0],6,$cont,"LR",0,"C",$fill);
        $this->Cell($w[1],6,utf8_decode(trim($row["cedula"])),"LR",0,"J",$fill);
        $this->Cell($w[2],6,utf8_decode(trim($row["nombre"])),"LR",0,"J",$fill);
        $this->Cell($w[3],6,utf8_decode(trim($row["apellido"])),"LR",0,"J",$fill);
        $this->Cell($w[4],6,utf8_decode(trim($row["login"])),"LR",0,"J",$fill);

        $this->Ln();
        $fill=!$fill;
        $cont++;
    }
    $this->SetX(15);
    $this->Cell(array_sum($w),5,'','T');
}
```

El método **Cuerpo()** es donde se realiza la consulta a la tabla o vista y se lista para mostrar cada registro del reporte.

Una vez que se ejecuta todo este proceso se carga una ventana externa



Para descargar el reporte en PDF, Clic en **Aceptar** y muestra el reporte en formato PDF.



REPUBLICA BOLIVARIANA DE VENZUELA

Fecha: 28/01/2010 **Hora:** 08:19:07 AM

REPORTE DE PERSONAS REGISTRADAS COMO USUARIOS

Nro	Cedula	Nombre	Apellido	Usuario
1	12345678	Admin	Admin	admin
2	19447744	Jesus	Bolivar	tembla

En el pie de la página en PDF también se muestra la siguiente información:

